

# 1 Enunciado

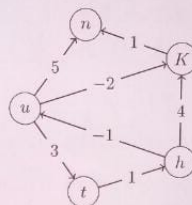
## ALGORITMOS Y ESTRUCTURAS DE DATOS III - 1<sup>er</sup> Parcial

Fecha examen: 08-MAY-2019 / Fecha notas: 22-MAY-2019

Completar:	Nº Orden	Apellido y nombre	L.U.	Cant. hojas <sup>1</sup>
	60	Braier Julian	531/17	9
No completar:	Nota (Nº)	Nota (Letras)	Docente	
	10	diez	Condino	

1. Utilizar el algoritmo de Ford para calcular los caminos mínimos desde el vértice  $u$  hacia todos los vértices del digrafo que aparece en la figura. Presentar pseudocódigo del algoritmo y realizar un seguimiento del mismo. Hecho esto, si se ordenan los vértices de acuerdo a su distancia al vértice inicial, podrá leerse el apellido del creador de  $\text{\LaTeX}$ , en el cual está basado  $\text{\LaTeX}$ .

2 p.



2. Un grafo funcional es un (pseudo) grafo dirigido en el cual todo vértice  $v$  cumple que  $d_{out}(v) = 1$ , de modo tal que representa a alguna función de un conjunto en sí mismo.

Sea  $G$  un grafo funcional.

- (a) Demostrar que los ciclos de  $G$  no comparten vértices.

0.75 p.

- (b) Demostrar que cada componente débilmente conexa de  $G$  tiene un único ciclo.

1.25 p.

SUGERENCIA: Existencia usando un ejercicio visto en clase. Unicidad usando el punto anterior.

3. Sea  $s$  una cadena de caracteres de longitud  $n$ . Sea  $d$  (diccionario) una lista de  $p$  cadenas de caracteres (palabras), cada una de longitud  $O(1)$ . Diseñar un algoritmo que decida si  $s$  puede partirse en subcadenas que están en  $d$ . Ejemplos:

2 p.

- $s = \text{"TikZistkeinZeichenprogramm"}$  puede partirse en subcadenas que están en  $d = \{\text{"TikZ", "Zeichenprogramm", "ist", "kein"}\}$ .
- $s = \text{"adorador"}$  puede partirse en subcadenas que están en  $d = \{\text{"ador", "noseusa"}\}$ .
- $s = \text{"adorador"}$  no puede partirse en subcadenas que están en  $d = \{\text{"adora", "dorador"}\}$ .

El algoritmo debe tener complejidad  $O(np)$  y estar basado en programación dinámica. Mostrar que el algoritmo propuesto es correcto y determinar su complejidad. Justificar.

SUGERENCIA: Resolver para cada prefijo de  $s$ .

4. Dados dos grafos  $G_1 = (V_1, E_1)$  y  $G_2 = (V_2, E_2)$ , se define su grafo junta como  $G_1 + G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup V_1 \times V_2)$ , es decir, el grafo que contiene a  $G_1$  y a  $G_2$  como subgrafos, y además contiene un eje entre cada vértice de  $G_1$  y cada vértice de  $G_2$ . Se dice que un grafo  $J$  es un grafo junta si y sólo si existen grafos  $G_1$  y  $G_2$  tales que  $J = G_1 + G_2$ .

2 p.

Sea  $G$  un grafo de  $n$  vértices y  $m$  ejes. Diseñar un algoritmo eficiente que decida si  $G = G_1 + G_2$  con  $G_1$  bipartito y  $G_2$  completo. Mostrar que el algoritmo propuesto es correcto y determinar su complejidad. Justificar. El mejor algoritmo que conocemos tiene complejidad  $O(m+n)$ , lo cual es necesario para obtener puntaje máximo en este ejercicio.

5. Un grafo (simple) se dice 1-árbol si es un árbol con un eje agregado.

2 p.

Sea  $G$  un grafo conexo con pesos asociados a sus ejes, y que tiene al menos un ciclo. Un 1-árbol generador de  $G$  es un 1-árbol que es subgrafo generador de  $G$ . Un 1-árbol generador mínimo de  $G$  es 1-árbol generador de  $G$  que tiene peso mínimo en el conjunto de los 1-árboles generadores de  $G$ .

Sea  $T$  un árbol generador mínimo de  $G$ , y sea  $e$  un eje de peso mínimo en el conjunto de ejes que están en  $G$  pero no en  $T$ . Demostrar que  $T + e$  es un 1-árbol generador mínimo de  $G$ .

SUGERENCIA: Demostrar que el peso de  $T + e$  es menor o igual que el peso de cualquier 1-árbol generador de  $G$ .

<sup>1</sup>Incluyendo a esta hoja. Entregar esta hoja junto al examen.

## 2 Resolución

### 2.1 Ejercicio 1

2

Braier 531/17  
Julian Algo III 1ª Parcial Hoja 1

Ej 1-

Función recursiva:

$$\hat{E}_i(u, v) = \begin{cases} 0 & \text{si } v = u \\ \infty & \text{si } v \neq u \wedge i = 0 \\ \min \{ \hat{E}_{i-1}(u, x) + c(x, v) \mid x \in N[v] \} & \text{otro caso} \end{cases}$$

↳ cuando que esto no vale si hay un ciclo neg. que contiene a  $u$ .

Donde  $\hat{E}_i(u, v)$  denote el camino más corto de  $u$  a  $v$  utilizando a lo sumo  $i$  aristas.

Pseudo código: asumo que tengo el grafo representado por lista de adyacencias.

Para todo nodo  $v$  en  $V(G)$ :  $\hat{E}_0(u, v) \leftarrow \infty$   
 $\hat{E}_0(u, u) \leftarrow 0$

Para  $i = 1, 2, \dots, n-1$ :

Para todo  $v \neq u$  en  $V(G)$ :

Para todo  $x \in N[v]$ :

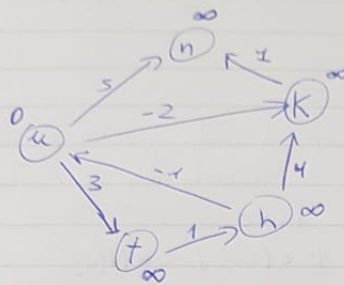
$$\hat{E}_{i-1}(u, v) \leftarrow \min \left( \hat{E}_{i-1}(u, v), \hat{E}_{i-1}(u, x) + c(x, v) \right)$$

$\left. \begin{matrix} \text{Para todo } v \neq u \text{ en } V(G) \\ \text{Para todo } x \in N[v] \end{matrix} \right\} O(m) \left. \vphantom{\begin{matrix} \text{Para todo } v \neq u \text{ en } V(G) \\ \text{Para todo } x \in N[v] \end{matrix}} \right\} O(n \cdot m)$

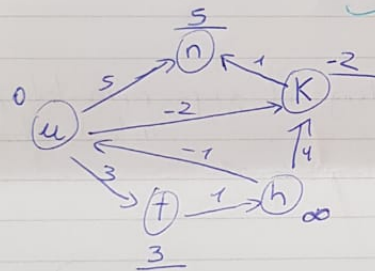
Observación: como sólo uso  $\hat{E}_i$  y  $\hat{E}_{i-1}$  puedo usar  $O(n+m)$  memoria en lugar de  $O(nm)$ .

✶ podías decir  $\nexists$  eje, pero está bien.

Paso 0: inicialización.

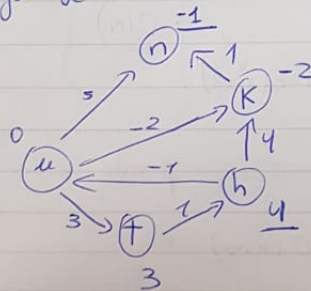


Luego de la 1<sup>ra</sup> iteración: tengo los caminos mínimos de longitud  $\leq 1$  *depende del orden de recorrida. En tu caso sí.*



(subrayo los valores que se modifican)

Luego de la 2<sup>da</sup>:



Braier 531/17

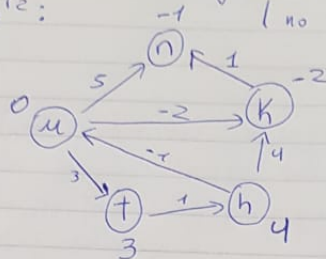
Julien

Algo III 1<sup>er</sup> Pasa

Hoja 2

Ej. 1.  
(continuación)

3<sup>er</sup>: (no hay cambios)



Después de la 4<sup>a</sup> no hay cambios nuevamente.

El apellido es Knuth. ✓

Dos comentarios:

• Una vez que se realiza una iteración que no produce cambios (en el caso del seguimiento en la tercera) no se producirán cambios en las siguientes. Puede optimizarse el algoritmo para que detecte esto y se detenga.

• El pseudo código que escribí precisa como precondición que no haya ciclos negativos. Puede modificarse para que detecte su presencia si los hay.



## 2.2 Ejercicio 2

2

Breier 531/17  
 Julián Algo III 1º Parcial Hoja 3

Ej 2-

a) Sean  $\{v_1, v_2, \dots, v_k, v_i\}$  y  $\{u_1, u_2, \dots, u_l, u_j\}$  dos ciclos simples de  $G$  tales que  $v_i = u_j$  (es decir, comparten este vértice). No pierdo generalidad al decir que es el primer vértice el que coincide.

✓ Para que ambos ciclos existan preciso que  $v_1 v_2 \in E(G)$  y que  $u_1 u_2 = v_1 u_2 \in E(G)$ . Como  $\deg(v_1) = 1$  esto sólo puede ser cierto si  $v_2$  y  $u_2$  son el mismo nodo.

✓ Puedo repetir el razonamiento para  $v_3$  y  $u_3$  y así continuar con el resto de los ciclos. Acabaré probando que si dos ciclos comparten un nodo entonces son el mismo ciclo, o sea, no hay dos ciclos diferentes que compartan nodos.

b) Sea  $C \subseteq V(G)$  una componente débilmente conexa. Propongo un algoritmo para encontrar un ciclo.

Conjunto de nodos visitados  $\leftarrow \emptyset$   
 Nodo actual  $\leftarrow u \in C$   
 mientras  $u \notin \text{visitados}$   
   agregar (visitados,  $u$ )  
    $u \leftarrow x \in N(u)$  // me muevo al único nodo que puedo ir desde  $u$

En castellano: elijo un nodo en  $C$  cualquiera. Como todos tienen grado de solito uno, tengo una única opción para seguir moviéndome. Voy avanzando y recordando los nodos que visité. Como no puedo salirme de  $C$ , que tiene cantidad finita de nodos, en algún momento

## 2.3 Ejercicio 3

Braier 531/17  
 Julián  
 Algo III 1er Parcial  
 Hoja 4

Ej 3-  
 Generalizo el problema:

puedePartirse( $s, i, d$ ) equivale a true sii los primeros  $i$  caracteres de  $s$  pueden partirse en subcadenas que estén en  $d$ . Defino recursivamente a esta función.

(\*) razonamiento  
 atrás

$$\text{puedePartirse}(s, i, d) = \begin{cases} \text{true} & \text{si } i = 0 \\ \exists s' \in d / \text{esSufijo}(s, s') \wedge \text{puedePartirse}(s, i - |s'|, d) & \text{si } i > 0 \end{cases}$$

, donde  $\text{esSufijo}(s, s')$  devuelva true sii  $s'$  es sufijo de  $s$ .  
 y  $|s'|$  denota la longitud de  $s' \in d$ . Como la longitud de las cadenas en el diccionario es  $O(1)$  asumo que  $\text{esSufijo}$  tiene igual complejidad.

En cada paso puedo hacer hasta  $p$  llamados recursivos. El peor caso sería si  $|s'| \equiv 1$  siempre. En ese caso la cantidad de llamados recursivos sería  $O(p^n)$ . Sin embargo, la cantidad de instancias recursivas es  $O(n)$  ya que sólo el parámetro  $i$  se reduce (asumo que  $\{s' \in d \mid |s'| \leq 1\}$ , tanto  $s$  como  $d$  no se modifican. Hay  $n+1$  valores posibles para  $i$ , desde  $n$  hasta  $0$ .  
 Conclusión, puede servir usar programación dinámica.

Utilizo como estructura de memorización al diccionario  $m$ , que por cada  $i$  entre 0 y  $n$  me devuelve la respuesta al problema si ya lo resolví o  $\perp$  si aún no lo hice.

*si me doy, si me te agrego un log.*

$m[0..n] \leftarrow \perp$

```

puedePartirse(s, i, d)
  if i = 0 return true
  if m[i] =  $\perp$ 
    definir(m, s, i, d)
  return m[i]
  
```

definir(m, s, i, d)

~~for j from 1 to p~~

int j ← 1  
 $j \leq p$   
 while ( ! esSufijo(s, d[j])  $\wedge$  puedePartirse(s, i - d[j], d) )  
 j++;

$1 \leq j \leq p+1$

(Invariante:  $\neg \exists k: N (1 \leq k < j \wedge \text{puedePartirse}(s, i - d[k], d))$ )

if j = p+1  $m[i] \leftarrow \text{false}$   
 else  $m[i] \leftarrow \text{true}$

*Complejidad  $O_2$  más otras*

Doy por hecho a la función bool esSufijo().  $O(n)$  ok.

Por resolver el problema que nos interesa hoy que llame ~~con~~ con  $i = n$ .

Braxer 531/17  
Julian

Algo III 1<sup>er</sup> Parcial

Hoja 5

Ej 3- (continuación)

⊗ razonamiento.

La cadena vacía siempre puede escribirse como una cadena de  $d$  (ninguna cadena).

Si no es vacía y ninguna en  $d$  es sufijo de  $s$  claramente no hay forma de escribir  $a$   $s$ . Supongamos que si hay una  $s'$  en  $d$  es sufijo de  $s$ . Sólo me vale servir esto si las primeras  $n-|s'|$  letras pueden partirse en cadenas de  $d$ .

⊗ Complejidad:

Tengo que resolver  $a$  lo sumo  $n$  veces recursivos. Al usar la estructura de memoización sólo tengo que computar cada caso una única vez, en las siguientes responderé  $O(1)$ .

Definir una array en  $m$  cuesta  $O(p)$ . Hago  $a$  lo sumo  $p$  iteraciones y en cada iteración hago operaciones que cuestan  $O(1)$ . Si ya está resuelto el caso  $m[i-1][j]$ . Puedo asumir que está resuelto ya que en este cálculo incluyo el costo de hacerlo. ✓

Conclusión: tengo que definir  $n$  arrays y cada uno me cuesta  $O(p)$ . Complejidad:  $O(np)$ . ✓

Muy claro!



## 2.4 Ejercicio 4

Nº Orden: / Fecha notas: 22-MAY-2019  
Apellido y nombre:

Brauer 531/17  
Julian  
Algo III 1er Parcial  
Hoja 6

Ej 4-

Observación: los nodos de  $G_2$  estén unidos a todos los otros nodos de  $G_2$  por ser  $G_2$  completo. También tienen una arista a todos los de  $G_1$  por ser  $G$  un grafo junta de  $G_1$  y  $G_2$ . Entonces los nodos están conectados a todos los otros de  $G$ , tienen grado  $n-1$ . ✓

Assumo que tengo a  $G$  representado por lista de adyacencias. Si viene como lista de aristas lo puedo transformar en  $O(nm)$ .

En  $O(nm)$  puedo encontrar a todos los nodos de  $G$  que tienen grado  $n-1$  (simplemente veo el cardinal de su vecindad). Elijo a los nodos que encuentre para formar parte de  $G_2$ . ⊕

Me falta ver que para el grafo inducido de  $G$  que queda al sacar los nodos de  $G_2$  que es bipartito. Vimos un algoritmo en la teórica para determinar si es bipartito en  $O(nm)$ . Consiste en obtener un árbol enraizado usando DFS o BFS y luego ver que en el grafo ~~no~~ no haya ninguna arista que ~~conecte~~ una (no al árbol)

dos nodos cuya profundidad en el árbol tenga la misma paridad. ✓

Si el grafo es no conexo hay que repetir para cada componente conexa. Si todas pueden ser bipartitas por separado entonces la unión de todas ellas es bipartita también. ✓

Algoritmo:

1.  $V(G_2) =$  todos los nodos que tienen grado  $= n-1$   $O(nm)$
2. obtener candidato a  $G_1$  filtrando a  $G$  para sacar lo que están en  $G_2$ .
3. chequear que  $G_1$  sea bipartito con el algoritmo de la teórica descrito arriba.  $O(nm)$

El algoritmo propuesto propone primero un  $G_2$  y luego se fija si lo que queda funciona como  $G_1$ , es decir, si es bipartito.  
¿Pero estoy dejando opciones válidas afuera al definir  $G_2$ ?

Claramente no puedo pasar nodos de  $G_1$  a  $G_2$ , ya que tener grado  $n-1$  es condición necesaria para estar en  $G_2$ .

Sólo que hacer al revés no me conviene (pasar nodos de  $G_2$  a  $G_1$ ) ya que si un grafo  $H$  es bipartito entonces  $H-v$  también lo es. O sea, no ayuda a que  $G_1$  cumpla la condición elegida.  
(esto sólo vale si  $|V(H)| \geq 3$ , después corrigo este error).

Entonces no, no dejo afuera posibilidades al definir  $G_2$  de esta forma. ✓

⊗ Esto no garantiza las cosas:

1.  $G_2$  es completo

2.  $G$  tiene todos los aristas que van de  $G_1$  a  $G_2$ .  
pueden ir

Después de consultar aprendí que tanto  $G_1$  como  $G_2$  deben tener nodos y que el grafo trivial no es bipartito. Eso desencadena algunas correcciones en el algoritmo.

1. Si  $n < 3$  devuelve falso ( $|V(G_1)| \geq 2$  por bipartito y  $|V(G_2)| \geq 1$ )

2. Si ~~no~~ ningún nodo tiene grado  $n-1$  (si  $|V(G_2)| = 0$ ) devuelve falso.

3. Si todos los nodos tienen grado  $n-1$  (si  $G = K_n$ ) devuelve verdadero. Porque puedo componer  $G_2$  con todos esos nodos menos dos ( $G_2 = K_{n-2}$ ) y, por lo tanto,  $G_1 = K_2$  que es bipartito. ✓

Braier 531/17  
Julia

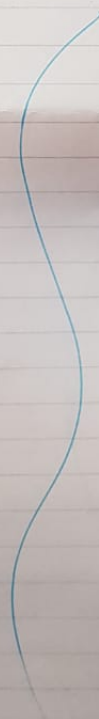
Algo III 1ª Parcial

Hoja 7

Ej 4- (continuación)

4. Si todos los nodos salvo 1 tienen grado  $n-1$  también devuelvo verdadero. Defino  $G_2 = K_{n-2}$  nuevamente. Esta vez, en consecuencia  $G_1 = \dots$  (grafo con dos vértices sin aristas), que también es bipartito.

Si ninguna de estas 4 condiciones se cumplen puedo trabajar con ~~siguiente~~ ~~antes~~. Ahora, con la garantía de que  $|V(G_2)| \geq 1$  y  $|V(G_1)| \geq 2$ .



## 2.5 Ejercicio 5

2

Braier 531/17  
Julián

Algo III 1<sup>a</sup> Parcial

Hoja 8

Ej 5- Cualquier árbol generado de  $G$  se puede escribir como  $T' + e'$  donde  $T'$  es un árbol y  $e'$  la arista extra.

estoy copiando los pesos  
↓

Tengo que  $T \leq T'$ , por ser  $T$  un AGM. Sin embargo, con lo que sé hasta ahora podría pensar que  $e > e'$ , en cuyo caso no podría garantizar que  $T + e \leq T' + e'$ .

Elegimos  $e$  por ser la arista de menor peso en  $G - T$ . Entonces si  $e' < e$  quiere decir que  $e' \in T$  (si no la habríamos elegido ella en lugar de  $e$ ). !

El árbol  $T' + e'$  tiene un único ciclo, que incluye a  $e'$  (visto a la teórica, es un árbol con una arista agregada). Llamo  $e''$  a la arista de peso máximo en este ciclo.

$T' + e' - e''$  es también un árbol (visto a la clase), lo llamo  $T''$ .

$e''$ , por ser la arista más pesada de un ciclo de  $G$ , sabemos que no pertenece al AGM  $T$  (visto a la clase). (En realidad podría ser que  $e''$  esté repetida con otra arista, que es lo que se quiere evitar del AGM. En ese caso elijo a la otra con  $e''$  y ya). !! O sea, tomamos una arista que seguro no está en  $T$ . Podría ser  $e'$  es un árbol.

Entonces tengo  $T' + e' = T' + e' - e'' + e''$  hacer eso directo  
 $= T'' + e''$  seg' es un árbol.

$T''$  es también un árbol, entonces  $T \leq T''$  ( $T$  AGM)

$e'' \notin T$ , entonces  $e \leq e''$  | si  $e'' < e$  sería falso que  $e$  es mínima en  $G - T$ .

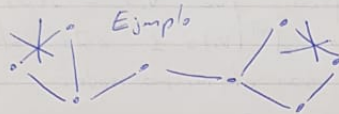
Ahora sí puedo ver que  $T + e \leq T' + e'$  por cualquier árbol  $T'$  y arista  $e' \notin T$  de  $G$ . O sea  $T + e$  es 1-AGM de  $G$ . ✓



voy a regresar a un nodo ya visitado. Todo lo que hice desde ese nodo en adelante conforma un ciclo. ✓

Falte ver que es único.

Supongamos una componente conexa con dos <sup>o más</sup> ciclos. Vi en la qe no computan vértices. Puedo sacar dos aristas, una de cada ciclo, y voy a seguir teniendo una componente conexa. (visto a dos).



Vimos en la teoría que una componente conexa tiene  $m \geq n-1$  aristas. En este caso particular, si aún sin dos aristas tengo algo débilmente conexo, sé que antes de quitar las aristas tenía  $|E(C)| \geq |V(C)| + 1$ , más aristas que nodos. Esto es imposible ya que dado que el  $\delta = 1$  para todos los nodos debería tener  $|E(C)| = |V(C)|$ . Absurdo. ✓

En conclusión: hay al menos un ciclo y menos de dos, o sea, hay un único ciclo en cada componente débilmente conexa.

Muy linda demostración !!